Complexidade Computacional de Módulos de Treliça de Códigos Convolucionais

Isaac B. Benchimol, Cecilio Pimentel, Richard Demo Souza e Bartolomeu F. Uchôa-Filho

***Resumo*—Neste artigo apresentamos uma medida de complexidade de decodificação de códigos convolucionais baseada em ciclos de máquina consumidos pela execução das operações aritméticas do algoritmo de Viterbi (VA) operando com decisão abrupta. Através da simulação destas operações definimos uma complexidade computacional do VA que mede de forma mais adequada a complexidade de treliça em esquemas de decodificação implementados por software. Implementações baseadas nos módulos de treliça convencional e mínimo são consideradas. Uma relação entre a complexidade definida neste trabalho e a complexidade de treliça definida por McEliece e Lin é investigada. A arquitetura adotada é a família de processadores digitais de sinais TMS320C55xx da Texas Instruments.**

***Palavras-Chave*—Códigos convolucionais, complexidade de decodificação, modulo de treliça, algoritmo de Viterbi.**

***Abstract*—We present in this paper a decoding complexity measure of convolutional codes in terms of machine cycles taken by the execution of the arithmetic operations of the Viterbi algorithm (VA) with hard decision decoding. By simulation of these operations we define a computational complexity of the VA that is a more adequate complexity measure if decoding is implemented by software. We conducted implementations based on both conventional and minimal trellis modules. A relation between the complexity defined in this work and the one stated by McEliece and Lin is investigated. The system architecture adopted is the TMS320C55xx digital signal processor family from Texas Instruments.**

***Keywords*—Convolutional codes, decoding complexity, trellis module, Viterbi algorithm.**

I. INTRODUÇÃO

Códigos convolucionais são amplamente utilizados em sistemas de comunicação digital por sua capacidade de aumentar a confiabilidade de transmissão [1]. Estes códigos podem ser representados por estruturas de treliça que permitem o uso eficiente de algoritmos de decodificação de máxima verossimilhança como o algoritmo de Viterbi (VA). Entretanto, o VA causa um grande impacto no consumo de energia do receptor. Em [2] são analisadas diferentes implementações de receptores para o padrão de redes sem fio IEEE 802.11 [3], mostrando que o VA contribui com 35% do seu consumo total. Este consumo está fortemente relacionado com a complexidade de decodificação do VA. Desta forma, buscar alternativas de decodificação menos complexas é essencial para sistemas de comunicação, sobretudo para os que têm severas limitações de energia.

Um código convolucional pode ser representado por várias treliças. Estas são estruturas periódicas, sendo o menor período denominado de módulo de treliça [4]-[6]. McEliece e Lin [4] definiram uma medida de complexidade de decodificação baseada no número total de bits que rotulam os ramos de um módulo de treliça (normalizado pelo número de bits de informação), denominada de complexidade de treliça de um módulo M, denotada por $TC(M)$. Um método para a construção do módulo de treliça mínimo foi apresentado em [4]. Este módulo tem uma estrutura irregular com número de estados em cada seção periodicamente variante no tempo em que várias medidas de complexidade de módulo de treliça são minimizadas. A busca de bons códigos convolucionais representados por treliças de baixa complexidade foi conduzida em [5]-[11], o que demonstra um grande interesse pelo tema.

O VA operando com decisão abrupta sobre M realiza operações aritméticas de soma e comparação de valores inteiros [4][12]-[14]. O número de somas por bit de informação é igual a $TC(M)$. Sendo assim, a $TC(M)$ representa a complexidade aditiva de M. Por outro lado, o número de comparações num estado específico de M é o número de ramos que converge para este estado menos um [13]. O número total de comparações em M representa a complexidade comparativa de M. A complexidade de treliça e a complexidade comparativa são duas medidas de complexidade do VA operando em um módulo M.

Neste trabalho propõe-se uma medida de complexidade, denominada de complexidade computacional de M, denotada por $TCC(M)$, que reflete mais adequadamente o esforço computacional de decodificação do VA executado por software. Para definir esta medida são considerados o número de somas e comparações, e seus respectivos custos (complexidades) computacionais de implementação. Estes custos são medidos em termos de ciclos de máquina consumidos pela sua execução. Desta forma, a $TCC(M)$ pode ser adotada como medida de complexidade do VA em aplicações em que o receptor seja implementado por software.

A família de processadores digitais de sinais TMS320C55xx da Texas Instruments (TI) foi utilizada como base deste trabalho. As implementações foram conduzidas através de simulações com linguagem C.

O restante deste artigo está estruturado da seguinte forma: na Seção II definimos o número de operações do VA; na Seção III determinamos o custo computacional de cada operação e

I. B. Benchimol, CMDI, Instituto Federal de Educação, Ciência e Tecnologia do Amazonas (IFAM), Manaus, AM, ibench@ifam.edu.br.

C. Pimentel, CODEC-DES, Universidade Federal de Pernambuco (UFPE), Recife, PE, cecilio@ufpe.br.

R. D. Souza, CPGEI, Universidade Tecnológica Federal do Paraná (UTFPR), Curitiba, PR, richard@utfpr.edu.br.

B. F. Uchôa-Filho, GPqCom-EEL, Universidade Federal de Santa Catarina (UFSC), Florianópolis, SC, uchoa@eel.ufsc.br.

Este trabalho recebeu apoio da FAPEAM e CNPq.

definimos a $TCC(M)$. Comparações entre a $TC(M)$ e a $TCC(M)$ são realizadas para códigos de diferentes taxas e com duas representações de treliça distintas: convencional e mínima. Na Seção IV apresentamos as conclusões.

II. COMPLEXIDADE DE MÓDULO DE TRELIÇA

Considere um código convolucional $C(n,k,v)$, em que v, k e n são o comprimento de restrição, o número de bits de entrada e o número de bits de saída, respectivamente. Em geral, um módulo de treliça M de um código convolucional $C(n,k,v)$ consiste em n' seções de treliça, 2^{v_t} estados na profundidade t, 2^{b_t} ramos conectando os estados entre as profundidades t e $t+1$, e l_t bits rotulando cada ramo entre as profundidades t e $t+1$, para $0 \le t \le n'-1$ [5]. Para realizar a decodificação usando o VA são necessários três componentes em cada seção da treliça: o calculador de distância de *Hamming* (HDC), o somar-comparar-armazenar (ACS) e o RAM *Traceback*. A seguir analisaremos as operações aritméticas requeridas pelo HDC e ACS sobre um módulo de treliça M para o VA operando com decisão abrupta. Como a operação do RAM *Traceback* não envolve operações aritméticas, este componente não será considerado neste trabalho.

A etapa de HDC consiste em calcular a distância de *Hamming* entre a palavra recebida e o bloco codificado em cada ramo de uma seção t do módulo de treliça M. Como cada ramo é rotulado com l_t bits, são necessárias l_t operações de comparação de bit cujos resultados são somados com $l_t - 1$ operações de soma. O número total de ramos no módulo é dado por $2^{v_t+b_t}$, portanto são necessárias $l_t 2^{v_t+b_t}$ operações de comparações de bit e $(l_t-1)2^{v_t+b_t}$ operações de soma. Definindo o número de operações de soma por S e o número de comparações de bit por C_b, concluímos que o número total de operações do HDC em uma seção t, T_t^{HDC}, é dado por

$$T_t^{HDC} = (l_t - 1)2^{v_t+b_t}(S) + l_t 2^{v_t+b_t}(C_b). \quad (1)$$

A etapa de ACS consiste em atualizar a métrica dos estados da treliça. Primeiramente o incremento de métrica de cada ramo é somado com a métrica de seu estado inicial, portanto são necessárias $2^{v_t+b_t}$ operações de soma. O próximo passo dessa etapa consiste em comparar as métricas acumuladas dos ramos que convergem para cada um dos estados da seção $t+1$ e selecionar a menor. Há $2^{v_{t+1}}$ estados na seção $t+1$ e $2^{v_t+b_t}$ ramos entre as seções t e $t+1$, portanto são comparados $2^{v_t+b_t}/2^{v_{t+1}}$ ramos por estado, que utilizam $(2^{v_t+b_t}/2^{v_{t+1}})-1$ operações de comparação. Considerando todos os estados, teremos $2^{v_t+b_t} - 2^{v_{t+1}}$ operações de comparação de valores inteiros. Definindo esta operação por C_i, concluímos que o número total de operações do ACS em uma seção t, T_t^{ACS}, é dado por

$$T_t^{ACS} = 2^{v_t+b_t}(S) + [2^{v_t+b_t} - 2^{v_{t+1}}](C_i). \quad (2)$$

A partir de (1) e (2) define-se o número total de operações do VA por bit de informação de um módulo de treliça M por

$$\begin{aligned} T(M) &= \frac{1}{k}\sum_{t=0}^{n'-1}(T_t^{HDC} + T_t^{ACS}) \\ &= \frac{1}{k}\sum_{t=0}^{n'-1} l_t 2^{v_t+b_t}[C_b + S] + (2^{v_t+b_t} - 2^{v_{t+1}})C_i. \end{aligned} \quad (3)$$

Considerando que a complexidade de treliça por bit de informação $TC(M)$ de um módulo de treliça M, para um código convolucional C de acordo com [4] é definida por

$$TC(M) = \frac{1}{k}\sum_{t=0}^{n'-1} l_t 2^{v_t+b_t} \quad (4)$$

e definindo a complexidade comparativa por bit de informação $MC(M)$ de um módulo de treliça M, para um código convolucional C por

$$MC(M) = \frac{1}{k}\sum_{t=0}^{n'-1}(2^{v_t+b_t} - 2^{v_{t+1}}) \quad (5)$$

em que $v_{n'} = v_0$, reescrevemos (3) usando (4) e (5) da seguinte forma

$$T(M) = TC(M)(C_b + S) + MC(M)C_i. \quad (6)$$

Para o módulo de treliça convencional, M_{conv}, temos $l_t = n$, $n' = 1$, $v_0 = v_1 = v$ e $b_0 = k$. Assim obtemos

$$TC(M_{conv}) = \frac{n}{k}2^{k+v} \quad (7)$$

$$MC(M_{conv}) = \frac{2^v(2^k - 1)}{k}. \quad (8)$$

O módulo de treliça mínimo é uma estrutura irregular com n seções que podem apresentar número de estados diferentes. Cada ramo é rotulado com apenas um bit [4]. Para este módulo de treliça mínimo, M_{min}, temos $n' = n$, $l_t = 1$ $v_t = \tilde{v}_t \ \forall t$, $b_t = \tilde{b}_t \ \forall t$ e $v_n = v_0$. Desta forma

$$TC(M_{\min}) = \frac{1}{k}\sum_{t=0}^{n'-1} 2^{\tilde{v}_t+\tilde{b}_t} \quad (9)$$

$$MC(M_{\min}) = \frac{1}{k}\sum_{t=0}^{n'-1}(2^{\tilde{v}_t+\tilde{b}_t} - 2^{\tilde{v}_{t+1}}). \quad (10)$$

Exemplo 1: Considere o código convolucional $C_1(7,3,3)$ com matriz geradora $G_1(D)$ dada por

$$G_1(D) = \begin{pmatrix} 1+D & 1+D & 1 & 1 & 0 & 1 & 1 \\ D & 0 & 1+D & 1+D & 1 & 1 & 0 \\ D & D & 0 & D & 1+D & 1+D & 1+D \end{pmatrix}.$$

As complexidades de treliça e comparativa do módulo de treliça convencional para $C_1(7,3,3)$ são $TC(M_{conv}) = 149,33$ e $MC(M_{conv}) = 18,66$. Portanto, obtemos de (6)

$$T(M_{conv}) = 149,33(S + C_b) + 18,66(C_i).$$

O módulo de treliça mínimo para $C_1(7,3,3)$ é mostrado na Figura 1. Definindo-se as complexidades de estados e de ramos de um módulo de treliça mínimo por $\tilde{\boldsymbol{v}} = (\tilde{v}_0, ..., \tilde{v}_{n-1})$ e $\tilde{\boldsymbol{b}} = (\tilde{b}_0, ..., \tilde{b}_{n-1})$, respectivamente, obtemos $\tilde{\boldsymbol{v}} = (3,4,3,4,3,4,4)$ e $\tilde{\boldsymbol{b}} = (1,0,1,0,1,0,0)$ para $C_1(7,3,3)$, o que implica em $TC(M_{min}) = 37,33$ e $MC(M_{min}) = 8$. Analogamente, obtemos de (6)

$$T(M_{min}) = 37,33(S + C_b) + 8(C_i).$$

Neste exemplo, o número relativo de operações requeridas pela treliça mínima em relação à treliça convencional é 25% para S e C_b e 42,8% para C_i.

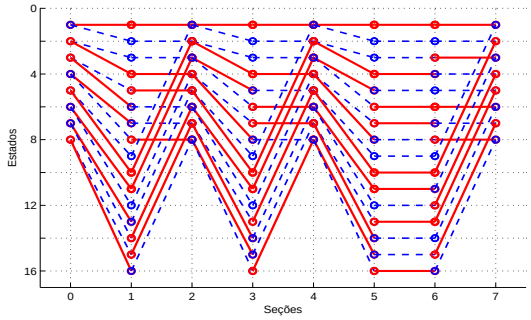

Fig. 1. Módulo de treliça mínimo para $C_1(7,3,3)$.

Uma vez determinado o número de operações do VA sobre um módulo de treliça, é necessário obter os custos individuais das operações S, C_b e C_i para que a comparação de complexidade seja mais significativa. Estes custos são determinados na próxima seção com base numa arquitetura particular.

III. COMPLEXIDADE COMPUTACIONAL DO VA

Nesta seção descreveremos a implementação das operações S, C_b e C_i para obter seus respectivos ciclos de máquina através de simulação da família de DSPs TMS320C55xx da TI. Estes processadores utilizam processamento com ponto-fixo e possuem desempenho otimizado, alta densidade de código e baixo consumo de energia [15]. Mais detalhes sobre estes processadores são encontrados em [16]. O ambiente de desenvolvimento integrado (IDE) utilizado foi o Code Composer Studio (CCStudio) v4 da TI [17].Uma vez obtidos estes valores, definiremos uma medida de complexidade computacional de um módulo de treliça.

A. *Implementação em DSP das operações do VA*

Operação soma (S)

A Tabela I mostra os detalhes da implementação da operação S. Foram usadas variáveis de tipo inteiro. Na primeira coluna aparece a operação soma de dois valores em linguagem C; na segunda, o código em linguagem *Assembly* do C55x gerado pelo compilador; na terceira, uma breve descrição do código, em que ACO.L é o registrador acumulador utilizado para receber o resultado da soma; na quarta, os respectivos ciclos de máquina. No total são necessários 3 ciclos de máquina para implementar a operação S.

Operação comparação de bits (C_b)

A operação C_b foi implementada com instruções lógicas XOR bit a bit, assumindo que cada bit da palavra recebida e da palavra-código em cada ramo do módulo da treliça tenha sido previamente armazenado em variável de tipo inteiro. A Tabela II mostra os detalhes da implementação da operação C_b. Na terceira coluna, ACO.L é o registrador acumulador utilizado para receber o resultado da operação XOR. No total são necessários 3 ciclos de máquina para implementar a operação C_b.

Operação comparação de inteiros (C_i)

A operação C_i foi implementada com uma instrução de seleção composta *if* incluindo o armazenamento do valor da menor métrica acumulada. A Tabela III mostra os detalhes da implementação da operação C_i. Na primeira coluna aparece a instrução *if* que compara duas métricas acumuladas, a menor é armazenada na variável de tipo inteiro *menor*. Na terceira, uma breve descrição do código explicada pelos seguintes passos: ACO.H e ACO.L são registradores acumuladores que recebem os valores das métricas acumuladas, aqui representadas pelas variáveis A e B, respectivamente. A seguir as métricas são comparadas, se B < A então o fluxo do programa é desviado para o rótulo @C1 e o valor de B é armazenado em *menor*. Seguindo esse caminho, o código consome 19 (1+1+16+1) ciclos de máquina. Caso A < B o código segue armazenando o valor de A em *menor* e desviando o fluxo do programa para o rótulo @C2 onde está a próxima instrução a ser executada. Um detalhe da arquitetura é que o valor em ACO.H não pode ser diretamente transferido para a memória, precisando ser copiado primeiro para ACO.L e depois então para a memória. Seguindo esse caminho, o código consome 17 (1+1+6+2+7) ciclos de máquina. Levamos em consideração o valor médio consumido pela operação, ou seja, 18 ciclos de máquina, justificado pela aleatoriedade dos valores. Em resumo, o custo computacional das operações do VA é mostrado na Tabela IV.

B. *Definição da complexidade computacional*

Substituindo-se os resultados obtidos em (6) definimos uma complexidade computacional de um módulo de treliça M, para esta arquitetura em particular, por

$$TCC(M) = TC(M)6 + MC(M)18. \qquad (11)$$

Observa-se que a $TCC(M)$ depende de duas medidas de complexidade do módulo M, a complexidade de treliça, que representa a complexidade aditiva, e a complexidade comparativa, sendo que o peso da última é o triplo do peso da primeira.

TABELA I. DETALHES DA IMPLEMENTAÇÃO DA OPERAÇÃO *S*.

Implementação em C	Assembly C55x	Descrição	Ciclos
S = code1 + code2;	LO(AC0) = *SP(#1) AC0.L = AC0.L + *SP(#0) *SP(#2) = AC0.L	AC0.L ← code1 AC0.L ← AC0.L + code2 S ← AC0.L	1 1 1
		Total	3

TABELA II. DETALHES DA IMPLEMENTAÇÃO DA OPERAÇÃO C_b.

Implementação em C	Assembly C55x	Descrição	Ciclos
Cb = code1 ^ code2;	LO(AC0) = *SP(#1) AC0.L = AC0.L ^ *SP(#0) *SP(#2) = AC0.L	AC0.L ← code1 AC0.L ← AC0.L XOR code2 Cb ← AC0.L	1 1 1
		Total	3

TABELA III. DETALHES DA IMPLEMENTAÇÃO DA OPERAÇÃO C_i.

Implementação em C	Assembly C55x	Descrição	Ciclos
If (A < B) menor = A; *else menor = B;*	HI(AC0) = *SP(#1) LO(ACO) = *SP(#2) cmp (ACO.H>=ACO.L) goto @C1 ACO.L = ACO.H goto C2@ || *SP(#3) = ACO.L @C1: *SP(#3) = ACO.L @C2: ... (próxima instrução)	ACO.H ← A ACO.L ← B se (ACO.H>=ACO.L) vá p/ @C1 ACO.L ← ACO.H vá para @C2 || menor ← ACO.L @C1: menor ← ACO.L @C2: ... (próxima instrução)	1 1 6 ou 16 2 7 1
		Total	17 ou 19

TABELA IV. CUSTO COMPUTACIONAL DAS OPERAÇÕES DO VA.

Operação	Ciclos
Soma S	3
Comparação de bit C_b	3
Comparação de inteiro C_i	18

Para o código $C_1(7,3,3)$ do Exemplo 1 obtemos $TCC(M_{conv}) = 1231{,}8$ e $TCC(M_{min}) = 368$. Logo, a complexidade computacional da treliça mínima é 30% da complexidade computacional da treliça convencional, mas em termos de complexidade comparativa, este percentual é 42,87% enquanto que para complexidade de treliça é 25%.

Nos próximos exemplos vamos analisar o impacto das complexidades de treliça, comparativa e computacional sobre códigos de taxas iguais.

Exemplo 2: Considere os códigos convolucionais $C_2(4,3,3)$ e $C_3(4,3,4)$ com matrizes geradoras $G_2(D)$ e $G_3(D)$ dadas por

$$G_2(D) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 0 & D & 1+D & 1 \\ D^2 & 1+D^2 & 1+D & 0 \end{pmatrix}$$

$$G_3(D) = \begin{pmatrix} D & D & 1 & 1 \\ 1+D & 1 & 0 & 1 \\ D+D^2 & 0 & D^2 & 1+D^2 \end{pmatrix}$$

e complexidades de estados e de ramos da treliça mínima dadas por $\tilde{\boldsymbol{v}} = (1,1,1,0)$ e $\tilde{\boldsymbol{b}} = (3,4,4,4)$ para C_2 e $\tilde{\boldsymbol{v}} = (1,0,1,1)$ e $\tilde{\boldsymbol{b}} = (4,4,3,4)$ para C_3. Nesse caso ambos os códigos apresentam $TC(M_{min}) = 32$, $MC(M_{\min}) = 13{,}33$ e $TCC(M_{\min}) = 431{,}94$. Entretanto, códigos com a mesma complexidade de treliça podem apresentar complexidade comparativa diferente. O próximo exemplo mostra essa situação.

Exemplo 3: Considere os códigos convolucionais $C_4(4,2,3)$ com perfis $\tilde{\boldsymbol{v}} = (3,2,3,4)$ e $\tilde{\boldsymbol{b}} = (0,1,1,0)$ e $C_5(4,2,3)$ com perfis $\tilde{\boldsymbol{v}} = (3,3,3,3)$ e $\tilde{\boldsymbol{b}} = (1,0,1,0)$, e matrizes geradoras $G_4(D)$ e $G_5(D)$ dadas

$$G_4(D) = \begin{pmatrix} D+D^2 & 1+D & D & 0 \\ D & 0 & 1+D & 1+D \end{pmatrix}$$

$$G_5(D) = \begin{pmatrix} D^2 & D & 1+D & 1+D \\ 1+D & 1+D & D & 1 \end{pmatrix}.$$

As complexidades de treliça, comparativa e computacional do módulo de treliça mínima de C_4 são respectivamente, $TC(M_{min}) = 24$, $MC(M_{min}) = 6$ e $TCC(M_{min}) = 252$. Para C_5 esses valores são $TC(M_{min}) = 24$, $MC(M_{min}) = 8$ e $TCC(M_{min}) = 288$. Embora ambos apresentem a mesma complexidade de treliça, isso não se reflete na complexidade comparativa gerando complexidades computacionais diferentes. Esse fato indica a importância de adotar a

complexidade computacional para comparar a complexidade de códigos convolucionais.

Exemplo 4: Considere os códigos convolucionais $C_6(4,3,4)$ com perfis $\tilde{\boldsymbol{v}} = (4,4,4,4)$ e $\tilde{\boldsymbol{b}} = (0,1,1,1)$ e $C_7(4,3,4)$ com perfis $\tilde{\boldsymbol{v}} = (4,5,4,4)$ e $\tilde{\boldsymbol{b}} = (1,0,1,1)$, e matrizes geradoras $G_6(D)$ e $G_7(D)$ dadas por

$$G_6(D) = \begin{pmatrix} D & D & D & 1+D \\ D & 1+D & 1 & 1 \\ D & D+D^2 & 1+D+D^2 & 0 \end{pmatrix}$$

$$G_7(D) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ D & D^2 & D+D^2 & 1 \\ D^2 & D+D^2 & 1+D & 0 \end{pmatrix}.$$

O código C_6 apresenta $TC(M_{min}) = 37,33$, $MC(M_{min}) = 16$ e $TCC(M_{min}) = 512$ enquanto que o código C_7 apresenta $TC(M_{min}) = 42,67$, $MC(M_{min}) = 16$ e $TCC(M_{min}) = 544$. Embora ambos apresentem a mesma complexidade comparativa, isso não se reflete nas complexidades de treliça e computacional. Observa-se ainda que o número de estados nas seções em que $\tilde{b}_t = 1$ é o mesmo para as duas treliças, porém o número total de estados é diferente.

IV. CONCLUSÕES

O objetivo deste artigo é apresentar uma medida de complexidade computacional de decodificação de códigos convolucionais baseada no VA operando com decisão abrupta. Mais precisamente, esta medida é relacionada com o número de ciclos de máquina consumidos pela decodificação. Isto foi alcançado determinando-se o número de operações aritméticas e seus respectivos custos computacionais de execução baseado numa plataforma de processadores digitais de sinais para módulos de treliça convencional e mínimo.

Neste estudo foram calculadas as complexidades de treliça, comparativa e computacional de códigos de várias taxas. No universo de códigos de mesma taxa, aqueles que apresentam a mesma complexidade de treliça podem apresentar complexidades computacionais diferentes. Portanto, a complexidade computacional proposta neste trabalho é uma medida mais fiel em termos de esforço computacional requerido pela decodificação implementada por software.

REFERÊNCIAS

[1] S. Lin and D. J. Costello, *Error Control Coding*, 2nd. edition, Prentice Hall, 2004.

[2] B. Bougard et al, "Energy-scalability enhancement of wireless local area network transceivers," *Proc. IEEE Workshop on Signal Processing Advances in Wireless Communications*, July 2004.

[3] IEEE Standard 802.11, "Wireless LAN medium access control (MAC) and physical (PHY) layer specifications: Hish spped physical layer in the 5 GHz band,"1999.

[4] R. J. McEliece, W. Lin, "The trellis complexity of convolutional codes", *IEEE Trans. Inform. Theory*, vol. 42, no. 6, pp. 1855-1864, Nov. 1996.

[5] B. F. Uchôa-Filho, R. D. Souza, C. Pimentel, M. Jar, "Convolutional codes under a minimal trellis complexity measure", *IEEE Trans. on Communications*, vol. 57, no. 1, Jan. 2009.

[6] H. H. Tang and M. C. Lin, "On (n,n-1) convolutional codes with low trellis complexity," *IEEE Trans. Commun.*, vol. COM-50, pp.37–47, Jan. 2002.

[7] I. E. Bocharova and B. D. Kudryashov, "Rational rate punctured convolutional codes for soft-decision Viterbi decoding", *IEEE Trans. Inform. Theory,* vol. 43, no. 4, pp. 1305-1313, July 1997.

[8] E. Rosnes and O. Ytrehus, "Maximum length convolutional codes under a trellis complexity constraint", *Journal of Complexity*, vol. 20, pp. 372-408, March-June 2004.

[9] B. F. Uchôa-Filho, R. D. Souza, C. Pimentel, and M.-C. Lin, "Generalized punctured convolutional codes", *IEEE Commun. Letters*, vol. 9, no. 12, pp. 1070-1072, Dec. 2005.

[10] A. Katsiotis, P. Rizomiliotis, N. Kalouptsidis, "New constructions of high-performance low-complexity convolutional codes", *IEEE Trans. on Communications*, vol. 58, no. 7, pp. 1950-1961, Jul. 2010.

[11] F. Hug, I. Bocharova, R. Johannesson, and B. D. Kudryashov, "Searching for high-rate convolutional codes via binary syndrome trellises", *in Proc ISIT* 2009, Seoul, Korea, 2009, pp. 1358-1362.

[12] A. Vardy, "Trellis structure of codes," *appeared in the Handbook of Coding Theory, V. Pless and W. Huffman, Elsevier*, pp. 1989–2117, 1998.

[13] R. J. McEliece, "On the BCJR trellis for linear block codes," *IEEE Trans. Inform. Theory*, vol. IT-42, pp. 1072–1092, July 1996.

[14] A. Lafourcade, A. Vardy, "Optimal sectionalization of a trellis," *IEEE Trans. Inform. Theory*, vol. IT-42, pp. 689–703, May 1996.

[15] S. M. Kuo, B. H. Lee, W. Tian, *Real-Time Digital Signal Processing: Implementations and Applications*, 2nd. Edition, Jonh Wiley & Sons, 2006.

[16] Texas Instruments, Inc., *TMS320C55x DSP CPU Reference Guide*, Literature no. SPRU371F, 2004.

[17] Texas Instruments, Inc., *Code Composer Studio user's guide (Rev B),* Literature no. SPRU328B, Mar. 2000.